LIBFREDO6

API and tips for Ruby Script Developers

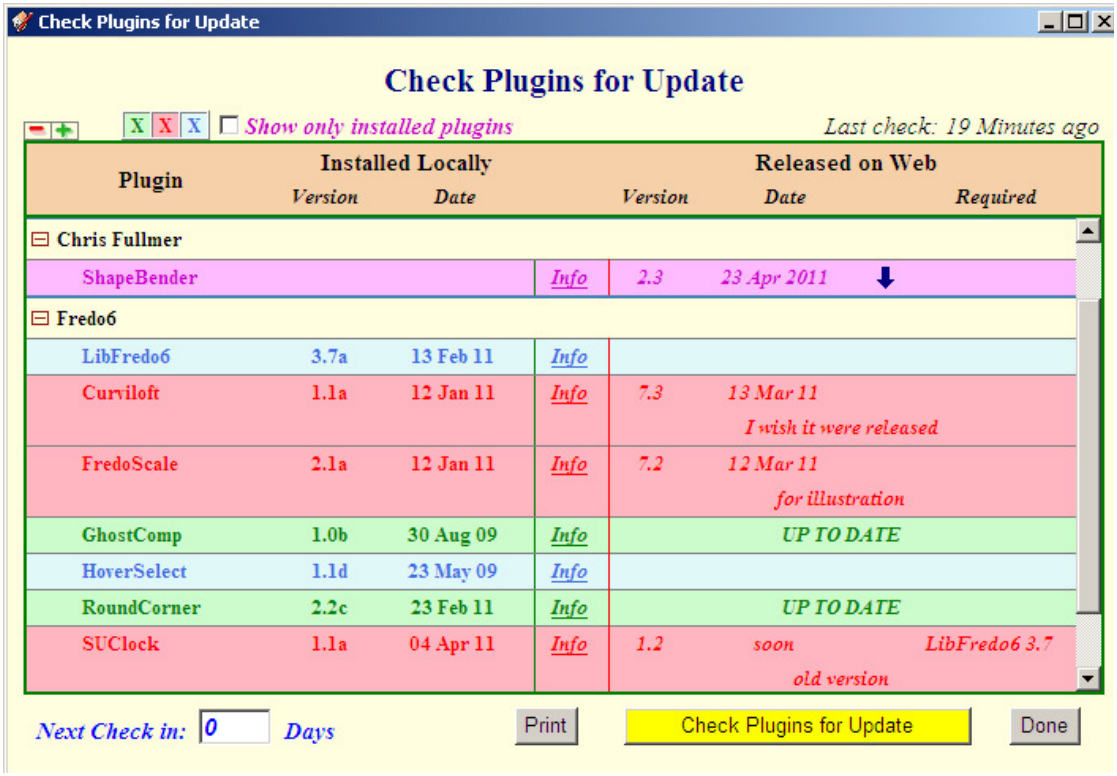
VERSION 3.7 – 17 APR 2011

LibFredo6 provides a few services that are open to scripts which are not developed with this library, namely:

1) **Participate in Check Plugins for Update**
2) **Register your Ruby script name for time tracking with SUClock**

1. Check Plugins for Update

As a reminder, the *Check Plugins for Update* dialog box matches information on plugins installed locally (on the left) with publishing information on the web (on the right).



The functionality available for script developers includes two independent parts:

a. Registering a plugin installed locally
b. Publishing information about the plugin on the web

a) Registering an installed Plugin

I opted for a passive method of registration because your script must work whether LibFredo6 is installed or not.

I assume that your script is embedded within a Module (or several modules), which is indeed a good practice.

All you have to do is to **define a method *register_plugin_for_LibFredo6* in one of your module**. This method just needs to return the plugin register information as a Hash Array, as described below.

Here is an example, assuming your module is "MyScriptModule" and you release a hypothetic plugin called **Yes We Can**:

```
def MyScriptModule.register_plugin_for_LibFredo6
   {
     :name => "Yes We Can",                  # Unique name for the plugin
     :author => "Barack",                    # Author name
     :version => "1.0a",                     # Version
     :date => "04 Nov 08",                   # Date of publication
     :description => "Force is with you",    # a short description
     :comment => "beta version",             # a short comment for the release
     :link_info => "www.ywc.com/plug",       # a URL where plugin info is posted
     :required => "LibFredo6 3.7",           # optional other plugin or library
   }
end
```

LibFredo6 will find your method and will register your plugin whenever the *Check Plugins for Update* feature is activated[1].

A few remarks on the Hash array returned by your method:

- All symbols should be defined in lowercase.
- All values are normally passed as Strings.
- For all field values, you can use language-dependent strings in one of the two forms (or combination of both):
  - "we can |FR| nous pouvons |DE| wir können"
  - ["we can", "|FR| nous pouvons", "|DE| wir können"]

  This applies to the ***:link_info*** URL as well, so that you can direct the user to different web pages depending on the current language.

  Note that it may not be a good idea to make the plugin name (in *:name*) language dependent.
- Not all fields are mandatory, although
  - if ***:name*** is not provided, then the registration is ignored
  - if ***:author*** is not provided it defaults to *"Anonymous"*
  - if ***:version*** is not provided, then there will be no comparison with a possible release on the web

[1] So there is no method activation at start up of Sketchup, but only when you call the dialog box

- For the purpose of checking for update, it is the field ***:version*** which is used for comparing releases, regardless of the *:date* field. If the field is in the form /\D*\s*(\d+\.*\d*)\s*(\w*)/i (like v3.7a or 2 or 2.0) then the comparison is made by taking into account the numeric pattern (so "v10.1a" is considered more recent than v9.0c". It is therefore strongly advised to use a compliant versioning scheme.
- **For dates**, it is recommended to use the notation `dd mmm yy` (like 02 Apr 11) so that it can be understood correctly by users from any country, without wondering about the order day, month, year. Note that dates are displayed as specified without any interpretation.
- Field ***:description*** and ***:required*** are displayed as tooltip of the plugin name.

Once you define the method ***register_plugin_for_LibFredo6***, you can check that your plugin definition appears in the *Check Plugins for Update* dialog box, in the left part.

Valid symbols are the following:

:name	**Name of the plugin**.
:author	**Author of the plugin**.
:version	**Version**. This is the field which is used to determine if the plugin is up to date or need upgrade.
:date	**Date of the release**. It is advised to use the format dd mmm yy
:description	**Short description of the plugin**
:comment	**A comment about the release**. Make it short!
:required	**A string containing the companion tool** (like LibFredo6 3.7). The string is displayed as-is in the right-most column of the Dialog box
:link_info	**A fully qualified URL for plugin information**.
:website **:url**	**[Optional] These parameters are relative to the web page where release information is published** *(see section b below)* ▪ **:website** → friendly name of the web site ▪ **:url** → qualified URL for the web page

I may include additional fields in a next release of LibFredo6.

b) Publishing Update information on the Web

The principle is to fetch the information from a web page, where the publishing information about the plugin is encoded in a special format, like for instance:

```
!!=!! Name = RoundCorner ; version = 2.2c ; Date = 23 Feb 11 ;
Author = Fredo6 ; comment = Stable version |FR| Version stable ;
required = LibFredo6 3.6!!=!!
```

The web page is designated by a website friendly name and by a qualified URL. By default:

- the ***website*** is "Sketchucation forum"
- the ***URL*** is "http://forums.sketchucation.com/viewtopic.php?f=180&t=3263" , which is a post in the Developer forum section where I publish the update information for my plugins.

You can however alter this default reference by registering locally your plugin via the module method `register_plugin_for_LibFredo6`, with the desired value of the fields ***:website*** and ***:url***.

For instance,

```
:website => "White House",          # just a friendly name of the web site
:url => "http://www.ywc.com/page=14"  # fully qualified URL of the publishing page
```

LibFredo6 reads the web page pointed to by the URL as plain text (the whole page) via the XMLHttpRequest mechanism and will find all strings with a specific pattern, as described below. So you can publish as many plugins information as you wish on the same page.

Here is how you can write the record which contains the publishing information related to your plugin:

- The record can be anywhere on the page
- The record must be enclosed between two delimiters **!!=!!**
- Symbol - value pairs are in the form **symbol = value**, with semi-column (**;**) used as a separator between pairs.

  !!=!! Name=Yes We Can ; version=2.0 ; Date= 04 Nov 12; author=Barack ;
  comment = Plugin name changed to Yes YOU can ; required = some good luck !!=!!
- The matching between an installed plugin and a we information record is done on the field [**name**]. For convenience however, the matching is case-insensitive (ex: JointPushPull will match joinpushpull).
- Don't use quotes or double quotes
- Space and new lines are ignored
- Symbol names are case-insensitive
- All HTML tags are also ignored. So Name=Yes We Can is possible and equivalent Name=Yes We Can[2].

[2] In most cases you write your page with a tool (like phpBB on Sketchucation forum, and you are in control of the resulting HTML formatting of the page).

The plugin information appears on the left (here in red, since the published version is more recent than the version installed locally):

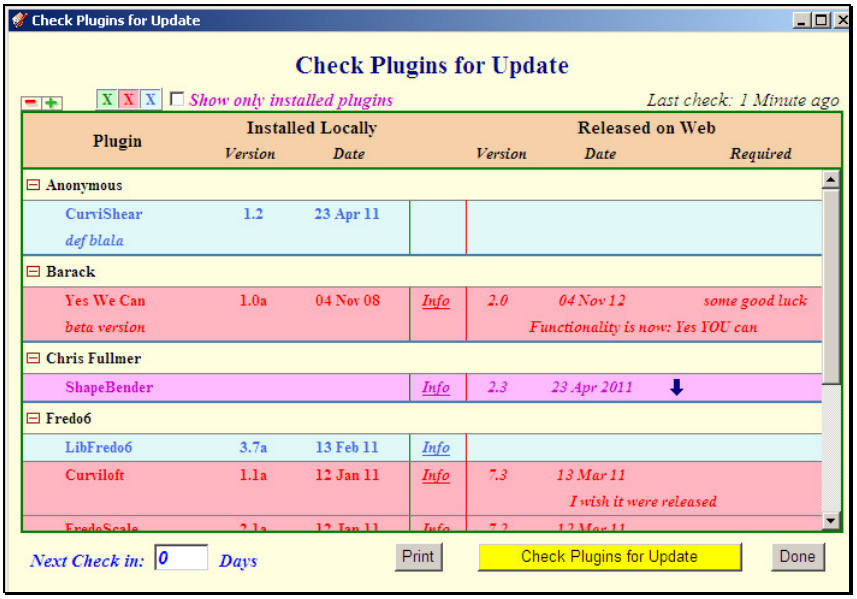
Barack						
Yes We Can	1.0a	04 Nov 08	Info	2.0	04 Nov 12	some good luck
beta version				Functionality is now: Yes YOU can		

The field names are basically the same as in the Plugin registration:

Name	**Name of the plugin**. It should be the same as the name registered locally
Author	**Author of the plugin**. It should be the same as the name registered locally
Version	**New version**. This is the field which is used to determine if the plugin is up to date or need upgrade.
Date	**Date of the new release**. It is advised to use a format dd mmm yy which is better understood internationally
Comment	**A short comment about the new published release**. Make it short!
Required	**A string containing the required companion tool** (like LibFredo6 or TTLib) The string is displayed as-is in the right-most column of the Dialog box. It is important to use this field when there is a change of the required version, which the user must be aware. It is also a good idea to publish the update information about companion library in the same place.
Info	**A fully qualified URL for plugin information**. This is optional, but allows you redirecting the user to a new page, different from the one registered locally

c) Additional remarks

1) The two processes, local registration and release information publishing, are independent. This means in particular that:
   - Users can see plugins published on the web that are not installed locally. This situation gets a violet color in the dialog box (assuming you unchecked the parameter *"Show only installed plugin"*.
   - Conversely, some plugins installed locally may have no release information published on the web (blue color):



2) The release information publishing is an **extra step** after you actually post the new release. It would have been more elegant to read directly the master post where you publish the plugin, but
   - LibFredo6 would have to check one page for each plugin, which would have significantly slowed down the check-for-update process (in addition, the first page of plugins main thread is often very long).
   - I think that in practice, it is better to control the upgrade publication for the users. You can publish the plugin on your main thread, wait to receive comments from active members, and then after a few days decide to publish it to all users.

3) I use a special thread on the Sketchucation forum to publish update information,

   http://forums.sketchucation.com/viewtopic.php?f=180&t=3263&p=15842#p15842

   It is however open to other script writers. Just create a response to the thread where you will publish the release information for all your plugins (so you really need only a single post).

   Although LibFredo6 check for update could check on various web sites, using a single thread (so a single page) will speed up the check process.

   I am however open to any other mechanism, in particular special pages, private to developers and without too much decoration and advertising.

   Last note: indeed this is not a thread to start a discussion. I put a notice in the first post.

4) The method I chose so far is not "silent". It requires a web dialog to be open in order to process the XmlHttpRequest read. It is possible however to perform it silently, without a dialog box, at least to inform the user that some installed scripts need an upgrade. I may do in a next release.

5) The missing steps would be indeed to provide a download and install automated procedure, in particular to avoid the frequent errors related to the unzip operation. I think I can manage it technically both on Windows and on Mac, but this requires a special access to the Sketchucation forum (files are not downloadable by program).

   In practice, I think this is not a major issue, because it is always good that users first read the main post about the plugin before deciding to download. Still, it could help some users if the installation and unzipping could be secured.

2. Registering Ruby name for Time Tracking with SUClock

In Sketchup, all Ruby interactive tools receive an identification number which is returned by the method ***active_tool_id*** of the Tools class. Unfortunately, this does not tell the name of the underlying plugin. Furthermore, the ID is not preserved across Sketchup sessions as it depends on how many other plugins and commands have been loaded. So the method ***active_tool_name*** invariably returns "RubyTool" whatever plugin is activated.

For SU Clock, you have the option to declare your Plugin name so that it can appear in clear text in the Tools statistics instead of the category *"Other Ruby tools"*.

⊟ Ruby Scripts		0:01:37	4.8%
JoinPushPull	*78.3%*	*0:01:16*	*3.8%*
RoundCorner	*11.5%*	*0:00:11*	*0.6%*
FredoScale	*4.2%*	*0:00:04*	*0.2%*
ToolsOnSurface	*2.0%*	*0:00:01*	*0.1%*
Other Ruby Scripts	*4.0%*	*0:00:03*	*0.2%*

Just insert the following line in the ***#activate*** method of your interactive Tool class:

```
def activate
  …
  LibFredo6.register_ruby "myscript" if defined?(LibFredo6.register_ruby)
  …
end
```

This method is very fast. It just creates an entry in an internal Hash array, associating the <name> argument to `Sketchup.active_model.tools.active_tool_id`. This is why it must be called in the `activate` step, not `initialize` method.

Also, be sure that you always include the test `if defined?(LibFredo6.register_ruby)`, because this method was not available in prior versions of LibFredo6 (and anyway, users may not have LibFredo6 installed at all).